## FROG 1

Có $N$ viên đá, được đánh số từ 1 đến $N$, viên đá thứ $i$ có độ cao là $h_i$. Có một con ếch đang ở vị trí viên đá 1, nó cần nhảy đến viên đá thứ $N$. Biết nó nhảy như sau: Nếu con ếch đang ở viên đá $i$, nó có thể nhảy đến viên đá thứ $i + 1$ hoặc $i + 2$ và mất $|h_i - h_j|$ sức với $j = i + 1$ hoặc $i + 2$.

Tìm cách nhảy cho ếch đến viên đá thứ $N$ mà mất tổng cộng ít sức nhất.

**Giới hạn:** các số đều là số nguyên

- $2 \leq N \leq 10^5$;
- $1 \leq h_i \leq 10^4$.

**Input**:

- Dòng đầu tiên là số $N$;
- Dòng thứ hai gồm $N$ số $h_i$.

**Output**:

In ra kết quả của bài toán.

| **INPUT** | **OUTPUT** | **Giải thích** |
|---|---|---|
| 4<br>10 30 40 20 | 30 | Nhảy theo thứ tự: 1 → 2 → 4 |
| 2<br>10 10 | 0 | Nhảy theo thứ tự: 1 → 2 |
| 6<br>30 10 60 10 60 50 | 40 | Nhảy theo thứ tự: 1 → 3 → 5 → 6 |

**Hướng dẫn giải:**

Nhận xét thấy: để nhảy đến vị trí thứ $i$, có thể nhảy từ vị trí thứ $i - 1$ hoặc từ vị trí $i - 2$.

Gọi $F[i]$ là tổng công sức ít nhất khi nhảy đến ô thứ $i$.

Ta có: $F[i] = min(F[i-1] + abs(h[i] - h[i-1]),\ F[i-2] + abs(h[i] - h[i-2]))$

Khởi tạo: $F[1] = 0$;

Kết quả: $F[N]$.

ĐPT: $O(N)$.

## FROG 2

Có $N$ viên đá, được đánh số từ 1 đến $N$, viên đá thứ $i$ có độ cao là $h_i$. Có một con ếch đang ở vị trí viên đá 1, nó cần nhảy đến viên đá thứ $N$. Biết nó nhảy như sau: Nếu con ếch đang ở viên đá $i$, nó có thể nhảy đến viên đá thứ $i+1, i+2, \dots, i+K$ và mất $|h_i - h_j|$ sức với $j$ là ô nhảy đến.

Tìm cách nhảy cho ếch đến viên đá thứ $N$ mà mất tổng cộng ít sức nhất.

**Giới hạn:** các số đều là số nguyên

- $2 \le N \le 10^5$;
- $1 \le K \le 100$;
- $1 \le h_i \le 10^4$.

**Input**:

- Dòng đầu tiên là số $N, K$;
- Dòng thứ hai gồm $N$ số $h_i$.

**Output**:

In ra kết quả của bài toán.

| INPUT | OUTPUT | Giải thích |
|---|---|---|
| 5 3<br>10 30 40 50 20 | 30 | Nhảy theo thứ tự: 1 → 2 → 5 |
| 3 1<br>10 20 10 | 20 | Nhảy theo thứ tự: 1 → 2 → 3 |
| 6 2<br>30 10 60 10 60 50 | 40 | Nhảy theo thứ tự: 1 → 3 → 5 → 6 |

**Hướng dẫn giải:**

Nhận xét thấy: để nhảy đến vị trí thứ $i$, có thể nhảy từ vị trí thứ $i-1, i-2, \dots, i-K$

Gọi $F[i]$ là tổng công sức ít nhất khi nhảy đến ô thứ $i$.

Ta có: $F[i] = min(F[i-j] + abs(h[i] - h[i-j]))$ với $j = 1 \dots K$

Khởi tạo: $F[1] = 0$;

Kết quả: $F[N]$.

ĐPT: $O(N \times K)$.

Code tham khảo:

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e5 + 10;
const int INF = 2*1e9;

int dp[MAXN], N, K, h[MAXN];

int main(){

    scanf("%d %d", &N, &K);
    for(int i = 1; i <= N; i++) scanf("%d", &h[i]);

    for(int i = 2; i <= N; i++){
        dp[i] = INF;
        for(int j = i-1; j >= max(i - K,1); j--)
            dp[i] = min(dp[i], dp[j] + abs(h[i] - h[j]) );
    }

    printf("%d\n", dp[N]);

    return 0;

}
```

## VACATION

Dino bắt đầu kì nghỉ hè từ ngày mai và anh ấy quyết định lập một kế hoạch cho mình. Kì nghỉ gồm $N$ ngày, mỗi ngày Dino có thể chọn một trong những hoạt động sau:

- A: bơi dưới biển – thu được $a_i$ điểm vui vẻ;
- B: leo núi – thu được $b_i$ điểm vui vẻ;
- C: học lập trình – thu được $c_i$ điểm vui vẻ;

Và Dino không muốn thực hiện hai hoạt động giống nhau trong hai ngày liên tục hoặc nhiều hơn. Hãy lên kế hoạch giúp Dino để có được tổng độ vui vẻ là lớn nhất.

**Giới hạn:** các số đều là số nguyên

- $1 \le N \le 10^5$;
- $1 \le a_i, b_i, c_i \le 10^4$.

**Input**:

- Dòng đầu tiên là số $N$;
- $N$ dòng sau, dòng thứ $i$ chứa ba số $a_i, b_i, c_i$ là điểm vui vẻ của các hoạt động trong ngày thứ $i$.

**Output**:

In ra kết quả của bài toán.

| INPUT | OUTPUT | Giải thích |
|---|---|---|
| 3<br>10 40 70<br>20 50 80<br>30 60 90 | 210 | $70 + 50 + 90$ |
| 7<br>6 7 8<br>8 8 3<br>2 5 2<br>7 8 6<br>4 6 8<br>2 3 4<br>7 5 1 | 46 | C, A, B, A, C, B, A. |
| 1<br>100 10 1 | 100 | |

**Hướng dẫn giải:**

Quy hoạch động trên bảng.

Nhận xét thấy: để xuống đến ô (i,j) thì có thể đi từ (i-1, k) với k khác j vì ô (i-1, j) "giống hoạt động".

Gọi $F[i,j]$ là tổng điểm vui vẻ lớn nhất khi xét đến ngày thứ $i$ và hoạt động thứ $j$.

Ta có: $F[i,j] = a[i,j] + max(F[i-1,k])$ với $k \ne j$

Khởi tạo:

Kết quả: $\max(F[N,j])$.

ĐPT: $O(N \times 3)$.

Code tham khảo:

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e5 + 10;

int N, a[MAXN], b[MAXN], c[MAXN], dp[MAXN][3];

int main() {
    scanf("%d", &N);
    for (int i = 1; i <= N; i++)
        scanf("%d %d %d", &a[i], &b[i], &c[i]);

    for (int i = 1; i <= N; i++) {
            dp[i][0] = max(dp[i-1][1],dp[i-1][2]) + a[i];
            dp[i][1] = max(dp[i-1][0],dp[i-1][2]) + b[i];
            dp[i][2] = max(dp[i-1][0],dp[i-1][1]) + c[i];
    }
    printf("%d\n", max(dp[n][0], max(dp[n][1], dp[n][2])));

    return 0;
}
```

## CÁI TÚI 1 & 2 – KNAPSACK 1

Có $N$ đồ vật, đồ vật thứ $i$ có trọng lượng là $W[i]$ và giá trị $V[i]$. Một tên trộm đột nhập vào siêu thị, tên trộm mang theo một cái túi có thể mang được tối đa trọng lượng $M$. Hỏi tên trộm sẽ lấy đi những đồ vật nào để được tổng giá trị lớn nhất.

Giải quyết bài toán trong các trường hợp sau:

- Mỗi vật chỉ được chọn một lần.
- Mỗi vật được chọn nhiều lần (không hạn chế số lần)

**Input:** file văn bản **BAG.INP**

- Dòng 1: $N, M$ cách nhau ít nhất một dấu cách.
- $N$ dòng tiếp theo: Mỗi dòng gồm hai số $W[i], V[i]$ là chi phí và giá trị đồ vật thứ $i$.

**Output:** file văn bản **BAG.OUT**: Ghi giá trị lớn nhất tên trộm có thể lấy

**Example:**

| BAG.INP | BAG.OUT |
|---|---|
| 5 15<br>12 4<br>2 2<br>1 1<br>1 2<br>4 10 | 15 |

**Giới hạn:**

- $N \leq 1000; M \leq 1000;$
- $W[i] \leq 1000;$
- $V[i] \leq 1000.$

**Hướng dẫn giải:**

1. Ta nhận thấy rằng: Giá trị của cái túi phụ thuộc vào 2 yếu tố: Có bao nhiêu vật đang được xét và trọng lượng còn lại cái túi có thể chứa được, do vậy chúng ta có 2 đại lượng biến thiên. Cho nên hàm mục tiêu sẽ phụ thuộc vào hai đại lượng biến thiên.

   Gọi $F[i, j]$ là tổng giá trị lớn nhất của cái túi khi xét từ vật 1 đến vật $i$ và trọng lượng của cái túi chưa vượt quá $j$. Với giới hạn $j$, việc chọn tối ưu trong số các vật $\{1, 2, \dots, i-1, i\}$ để có giá trị lớn nhất sẽ có hai khả năng: xét tại vật thứ $i$:

   - Nếu không chọn vật thứ $i$: tức là không có vật thứ $i$ thì trọng lượng $i-1$ vật trước vẫn là $j$:
   $$F[i, j] = F[i-1, j]$$
   - Nếu chọn vật thứ $i$:

     **Trường hợp mỗi vật được chọn 1 lần:** giờ mới chọn thêm vật $i$ vào, trước đó chỉ có $i-1$ vật và trọng lượng trước khi chọn vật thứ $i$ là $j - W[i]$. Khi đó ta có giá trị lớn nhất sẽ bằng giá trị là:
     $$F[i, j] = V[i] + F[i-1, j - W[i]]$$
     Với $V[i]$ là giá trị của vật thứ $i$.

     Ta có **công thức truy hồi** như sau:

     - $F[0, j] = 0$ (hiển nhiên) – Bài toán con nhỏ nhất.
     - $F[i, j] = max(F[i-1, j], V[i] + F\big[i-1, j - W[i]\big]).$

     **Trường hợp mỗi vật được chọn nhiều lần:** giờ chọn thêm vật $i$ vào nữa, trước đó có thể có vật i rồi. Vậy trước đó chỉ có $i$ vật và trọng lượng trước khi chọn thêm vật thứ $i$ là $j - W[i]$. Khi đó ta có giá trị lớn nhất sẽ bằng giá trị là:
     $$F[i, j] = V[i] + F[i, j - W[i]]$$

Với $V[i]$ là giá trị của vật thứ $i$.

Ta có **công thức truy hồi** như sau:

- $F[0, j] = 0$ (hiển nhiên) – Bài toán con nhỏ nhất.
- $F[i, j] = max(F[i-1, j], V[i] + F[i, j - W[i]])$.

2. Kết quả: $F[N, M]$
3. ĐPT: $O(N * M)$
4. Truy vết:

**Trường hợp 1**: $F[N, M]$ là giá trị lớn nhất thu được khi chọn trong cả $N$ vật với giới hạn trọng lượng là $M$.

- Nếu $F[N, M] = F[N-1, M]$ thì tức là không chọn vật thứ $N$, ta truy về $F[N-1, M]$.
- Còn nếu $F[N, M] \neq F[N-1, M]$ thì ta thông báo rằng phép chọn tối ưu có chọn vật thứ $N$ và truy về $F[N-1, M - W_n]$.

```
void truyVet(int N, int M){
   if (F[N, M] == 0) return;
   if (F[N, M] == F[N-1, M]) truyVet(N-1, M);
   else {
      cout << N << ' ';
      truyVet(N-1, M-W[N]);
   }
}
```

**Trường hợp 2:** $F[N, M]$ là giá trị lớn nhất thu được khi chọn trong cả $N$ vật với giới hạn trọng lượng là $M$.

- Nếu $F[N, M] = F[N-1, M]$ thì tức là không chọn vật thứ $N$, ta truy về $F[N-1, M]$.
- Còn nếu $F[N, M] \neq F[N-1, M]$ thì ta thông báo rằng phép chọn tối ưu có chọn vật thứ $N$ và truy về $F[N, M - W_n]$.

```
void truyVet(int N, int M){
   if (F[N, M] == 0) return;
   if (F[N, M] == F[N-1, M]) truyVet(N-1, M);
   else {
      d++;
      if (F[N, M-w[N]] == F[N-1, M-W[N]]){
         res.push_back(make_pair(N, d)); // vật N được chọn d lần
         d = 0;
      }
      truyVet(N, W-W[N]);
   }
}
```

## CÁI TÚI 3 – KNAPSACK 2

Tương tự cái túi 1 và 2 nhưng với giới hạn:

**Giới hạn:**

- $N \le 100; M \le 10^9;$
- $W[i] \le M;$
- $V[i] \le 1000.$

Vậy khi này thuật toán với ĐPT $O(N * M)$ không thỏa mãn nữa.

Ta sẽ nhận xét để định nghĩa lại hàm mục tiêu: dựa trên số có bao nhiêu vật được xét và giá trị đang thu được là bao nhiêu.

Gọi $F[i, j]$ là <u>tổng trọng lượng tối thiểu</u> khi <u>xét từ vật 1 đến vật $i$</u> và <u>tổng giá trị các vật trong túi là $j$</u>. Xét tại vật thứ $i$, có hai khả năng:

- Nếu không chọn vật thứ $i$:
$$F[i, j] = F[i-1, j]$$
- Nếu chọn vật thứ $i$:
$$F[i, j] = W[i] + F[i-1, j - V[i]]$$

Ta có **công thức truy hồi** như sau:

  - $F[0, j] = INF; F[0,0] = 0$
  - $F[i, j] = min(F[i-1, j], W[i] + F\left[i-1, j - V[i]\right]).$

Kết quả: $res = F[N, \max(j)] \le M$

ĐPT: $O(N * (N * V))$

## LCS

Cho hai xâu $S$ và $T$. Tìm xâu con chung dài nhất của hai xâu đó. Xâu A được gọi là xâu con của xâu B khi xóa một số kí tự của xâu B đi thì được xâu A.

**Giới hạn:** các kí tự là các chữ cái thường và độ dài xâu không quá 3000.

**Input**:

- Gồm hai dòng là xâu $S$ và $T$.

**Output**:

In ra xâu con chung dài nhất.

| INPUT | OUTPUT | Giải thích |
|---|---|---|
| axyb<br>abyxb | axb | ayb cũng được chấp nhận. |
| aa<br>xayaz | aa | |
| a<br>z | | |
| abracadabra<br>avadakedavra | aaadara | |

**Hướng dẫn giải:**

Ta có thể gọi $F[i,j]$ là độ dài dãy con chung dài nhất xét khi xét $i$ kí tự đầu tiên của xâu $S$ và $j$ kí tự đầu tiên của xâu $T$.

- Nếu $S[i] = T[j]$ thì $F[i,j] = F[i-1,j-1] + 1$;
- Ngược lại, $F[i,j] = \max(F[i-1,j], F[i,j-1])$.

Khởi tạo:

Kết quả: $F[M,N]$.

ĐPT: $O(M \times N)$.

Code tham khảo:

```
string a, b, n, m;
int f[MAXN][MAXN];

int lcs() {
    a = ' ' + a;
    b = ' ' + b;

    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            if (a[i] == b[j]) f[i][j] = f[i-1][j-1] + 1;
            else f[i][j] = max(f[i-1][j], f[i][j-1]);

    return f[m][n];
}

void result(){
    string res = "";
    while (m && n)
        if(a[m - 1] == b[n - 1]){
            res += a[m - 1];
            m--;
            n--;
        }
        else if(dp[m - 1][n] > dp[m][n - 1]) m--;
            else n--;

    reverse(res.begin(), res.end());
    cout << res;
}
```